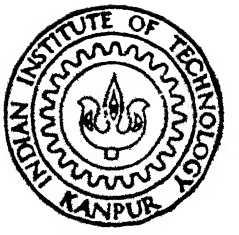
REFERENCE AND ELLIPSIS IN A NATURAL LANGUAGE INTERFACE TO DATABASES

By

Y. KRISHNA BHARGAVA



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MAY, 1992

REFERENCE AND ELLIPSIS IN A NATURAL LANGUAGE INTERFACE TO DATABASES

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
Y. KRISHNA BHARGAVA

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MAY, 1992

Acknowledgements

I wish to thank my supervisor, Dr. Rajeev Sangal for his guidance and help throughout the course of this thesis. I also thank Dr. Vineet Chaitanya for his enthusiastic support and valuable suggestions. I would be happy, if I can have a part of their unbounded zeal.

My thanks are due to the NLP and MT groups who gave me a feel of collective activity and also helped me during my thesis.

Special thanks to Ravindra, who is always helpful in his genial manner and the H-top junta. Lu and Sinha provided the much needed data in developing the system. Many thanks to all those friends who made my stay at the institute a pleasant experience.

ABSTRACT

In order for a natural language interface to be useful it must accept anaphora, ellipsis and other means of abbreviating utterances. Methods to handle these are designed and implemented in a prototype NLI for Hindi. The Paninian parser developed at IIT Kanpur is used for forming the parse structure of the question. The E-R model is used to capture the structure of the database. A database independent intermediate representation of the question is formed. Algorithms for mapping from parse structure to intermediate representation and translating from intermediate representation to the formal query language are developed. The modular structure of the NLI makes it easily adaptable to new databases, applications, or other Indian languages.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. OVERVIEW OF NLI 7

2.1 Modules of the NLI 8

2.1.1 Case for Modular Approach 8

2.2 Paninian Parser 9

2.2.1 Morphological Analyzer 11

2.2.2 Local Word Grouper 12

2.2.3 Core Parser 12

2.3 Extensions to the Parser 14

2.3.1 Value Recognition and Unknown Terms 14

2.3.2 Pronouns and Definite Noun Phrases 15

2.3.3 Ellipsis Handler 15

2.3.4 Logical Divison of Lexcon 15

3. MAPPING 16

3.1 Domain Knowledge 16

3.1.1 Knowledge Representation in Our System 17

3.2 Intermediate Representation 19

3.3 Mapper 21

3.3.1 First Phase 22

3.3.2 Second Phase 25

3.3.3 Third Phase 30

3.4 Translator 31

3.4.1 Target Language 32

3.4.2 Code Generation for Directed Tree 33

3.4.3 Code Generation for a Node 36

3.5 Implementation 37

3.6 Accessing the Database 37

4. REFERENCE AND ELLIPSIS 38

4.1 Ellipsis 38

4.1.1 Surface Level Ellipsis 38

4.1.2 Deep Level Ellipsis 41

4.2 History List Formation 42

4.2.1 Algorithm 43

4.3 Reference 43

4.3.1 Anaphora 43

4.3.2 Others 48

4.4 Domain Knowledge in Resolution 49

5. CONCLUSIONS 50

REFERENCES 53

APPENDIX 1 56

APPENDIX 2 58

APPENDIX 3 59

LIST OF FIGURES

Fig. 2.1 Overall structure of the NLI 7

Fig. 2.2 Stucture of the parser 11

Fig. 3.1 E-R diagram for the library domain 18

Fig. 3.2 Structure of a node 19

Fig. 3.3 An intermediate representation 20

Fig. 3.4 A sample map table 23

Fig. 3.5 Set of nodes returned by Get_ER_Info 23

Fig. 3.6 Output of Construct_Arcs 29

Fig. 3.7 A sample translator table 33

Fig. 3.7 Generated SQL query 34

CHAPTER 1

INTRODUCTION

Natural language has always been considered the key to bridge the gap between a casual user and a complex system. In fact, the first difficulty one encounters in approaching an unknown machine and trying to use it, arises from the fact that the machine speaks a language different from that of the user: he cannot talk to it in his native language and he cannot expect to engage in a friendly dialogue. Though much effort has been devoted to the study of communication between man and machine, with some of the experimental systems going into actual use, many problems in this area remain to be solved. The problem of natural language communication is further accenuated in the Indian context, where there are many different languages, since a cost effective way of supporting all the languages, is an additional difficulty.

1.1 Data Base Management Systems

Data Base Management Systems (DBMS) provide means for building and maintaining large, shared databases based on computational theories of information for efficient processing. With computers becoming cheaper and widely available more and more data is being computerised using the DBMS.

The community of potential information system users is growing rapidly with advances in hardware and software tech-

nology that permit computer/communications support for more and more application areas. Some of the many areas where computerisation is being done rapidly are railway reservations, management information systems, library services, university admissions and courses etc. So the question of providing access to this information in a easy manner is to be addressed.

1.2 Do Databases need Natural Language?

Natural languages are the ultimate knowledge representation languages. Most of the human knowledge is written in books in natural languages. Everything that people can express in any formal language can be expressed in any natural language, usually more readably. Their major weakness is the computational complexity of processing them. Yet natural languages serve as the standard for semantics: any formalism that is rich enough to handle ordinary language should be rich enough to handle anything.

All the major semantic features of natural languages can be useful for database systems [SOW90]. They include

* Declarative sentences for expressing and updating facts,
* Interrogative sentences for questions,
* Imperative sentences for commands and procedures,
* Nouns, verbs, and adjectives for referring to things, actions, states, and properties,
* Numbers and plurals for sets and counting,

* Tenses and temporal adverbs for time sequences,
* Modalities for constraints, options, and requirements,
* Conjunctions for logical connectives,
* Quantifying words to refer to sets of entities,
* Prepositions and case endings for relationships,
* Meta language for explanations and help,
* Metaphor as a mechanism for learning and extensibility.

But there is also a possibility that natural language is richer than necessary. Perhaps database systems do not require as rich a formalism as the systems developed for natural language semantics. So, systems that do not require the full power could implement a subset and leave the full language as a target for future growth.

1.2.1 Natural Language Interfaces

The most common application of natural language systems has been in the construction of natural language interfaces to database systems. These systems offer the simplest view of context: there is a fixed database of knowledge that does not change, and the interpretation of any given sentence is virtually independent of the interpretation of the preceding sentences. As such the natural language is viewed simply as convenient query language for the database, and the system's only task is to map sentences into an actual database query.

A major benefit of using natural language is that it shifts onto the system the burden of mediating between two

views of the data -- the way in which it is stored (the "database view") and the way in which an end user thinks about it (the "user's view"). Basically, database access is done in terms of files, records, and fields, while natural language expressions refer to the same information in terms of entities, the attributes of these entities, and the relationships between them in the world.

1.2.2 Background

In the past many natural language interfaces were limited by the lack of suitable database support [BOL86]. Many systems used their own filing scheme to overcome this problem. Also many of the systems require substantial amount of work, comparable to the effort involved in developing the original system, to transport them to a new DBMS or a new domain. The problem of providing useful responses in case of null values in database needs to be fully addressed.

While there is much work going on for English, there is little work on Indian languages to make them available on computer. The work at IIT Kanpur [BHR90b] indicates that a more efficient framework can be used for parsing Indian languages. The Paninian parser developed based on this theory can be adapted to all Indian languages with minor or no changes.

The issue of database support is not a problem in the present day, because relational databases are widely avail-

able with at least one query language, usually SQL, provided.

1.3 Scope of Problem

The development of an NLI should not be on an ad hoc basis but should focus on the underlying problems that affect its performance. The following important issues need to be addressed in an NLI :

- Reference, ellipsis, and quantification
- Domain knowledge representation
- Intermediate representation and mapping to formal query language
- To keep the structure of the NLI in such a way that it is easily adaptable to new applications, databases, or even different natural languages.

We make a technical distinction between the words "question" and "query". A question is any string entered by the user to the NLI. A query is a formal representation of a question in the query language of a DBMS. The type of NLI envisioned is targeted at the user who is unfamiliar with the particular system he needs to use. It is assumed that the user has some understanding of the underlying task.

1.4 Reference and Ellipsis

The issues of anaphora, ellipsis, and definite noun

phrases in natural language interfaces are addressed. Algorithms to handle them are designed and are implemented in a prototype NLI.

1.5 Capturing Domain Knowledge

Domain knowledge is captured in an E-R model for the domain. This is used as an information source detached from the actual modules.

1.6 Intermediate Representation and Mapping

An E-R based intermediate representation is developed. Algorithms for mapping from parse structure to intermediate representation and translating from intermediate representation to SQL query are developed.

1.7 Outline of the Thesis

In the second chapter the flowchart of the NLI is presented and the structure of the Paninian parser is described. Chapter 3 discusses the mapper and translator modules of the NLI. In the next chapter the issues of reference and ellipsis are discussed and the methods to handle them are presented. In Chapter 5 conclusions and possible extensions are listed.

CHAPTER 2

OVERVIEW OF NLI

The flow diagram of the NLI is shown in Fig. 2.1.

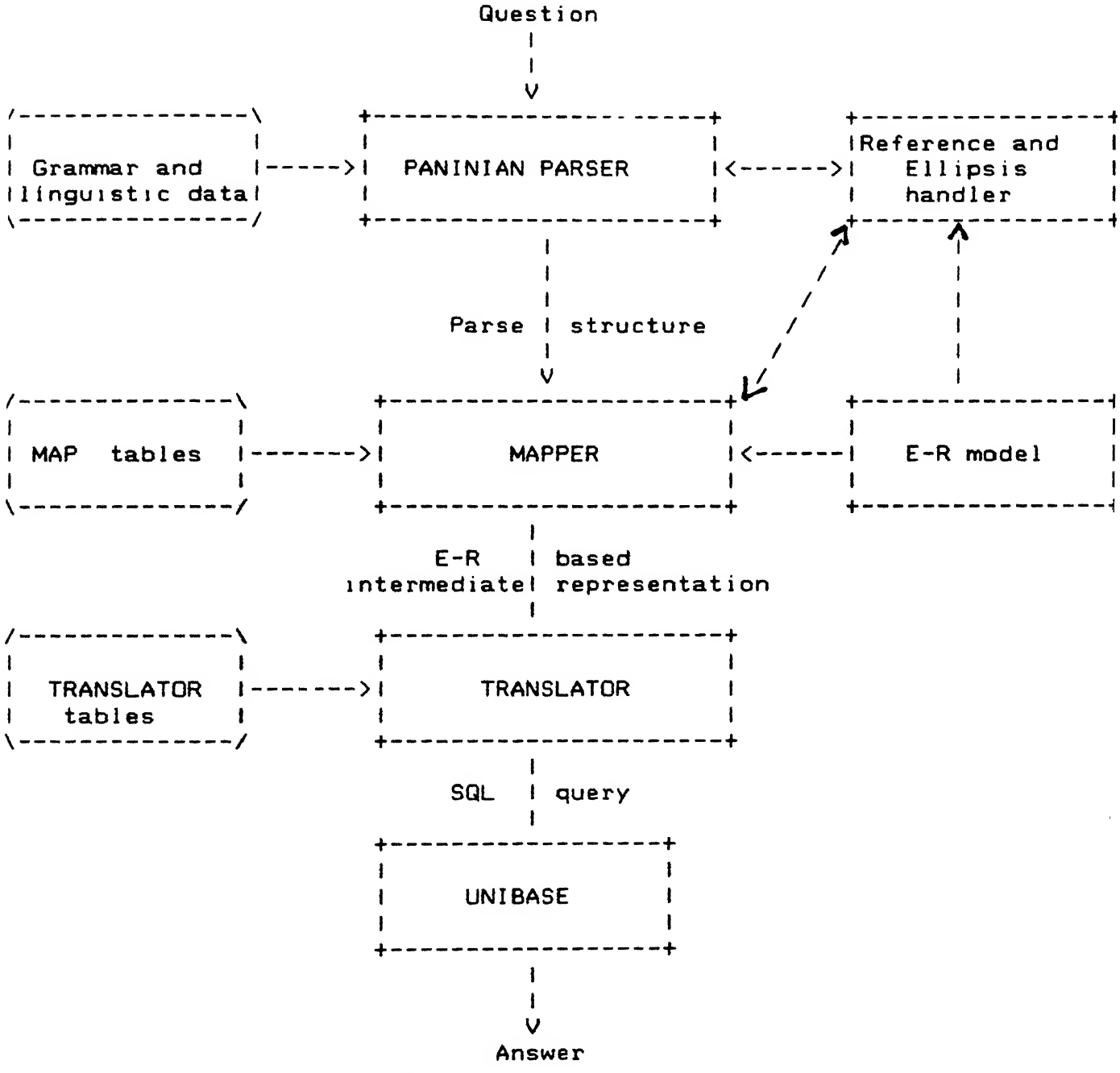


Figure 2.1. Overall structure of the NLI

2.1 Modules of the NLI

In Fig. 2.1, parser, reference and ellipsis handler, mapper, and translator are the processing modules while the other blocks are the information sources. The detailed functioning of each module is presented in the ensuing sections.

2.1.1 Case for Modular Approach

Modularity is an important feature of successful systems. There are many advantages for a modular NLI, exploiting the various bodies of knowledge required to interpret questions in successive stages of processing.

1. Carrying out syntactic processing of a question before semantic analysis so that the syntactic structure of the question can be used to guide semantic operations.

2. Easier to control the various subprocesses involved in a modular system.

3. Easier to localise and define types of error, and to provide relevant information about them.

4. Transportability to a different Language, domain, and DBMS can be easily achieved.

Each module builds a representation of the question in an appropriate formal structure, the final representation

being the SQL query. The modules are quite general. Only the information sources need be changed to achieve any kind of independence wanted.

The parser module can parse another language, simply by changing the grammar and the linguistic data, which are the information sources for this module. This module need not be changed for a different domain or a different DBMS. The mapper module can be used for a different domain, by changing the E-R model which captures the domain knowledge. This module need not be changed for a different database in the same domain. The translator module can be used for a different database by changing the database schema tables which reflect the actual structure of the database.

But there is also some cost associated with this modular approach. The necessary links between various forms of question representation are to be clearly specified and may be a overhead.

2.2 Paninian Parser

The parse structure is centered around the verbal groups of the sentence and is based on the karaka relations. The karaka relations are syntactico-semantic relations between the verbal groups and the nominal groups in the sentence. The parse structure is a mapping between the nominals and the karaka relations. Besides the karaka relations the

sentence may contain the non-karaka relations such as those contributed by the adjectival relations, purpose and relational words. The karaka relations are used for the disambiguation of word senses. Thus the parse structure, contains a mapping between the nominals and the karaka relations along with the concepts of the various words in the sentence.

The parsing process makes use of two components, the parser program and the grammar of the language. The parser program forms the procedural part and the grammar of the source language forms the declarative part of the parser. The grammar changes depending on the language to be parsed, while the program remains unchanged. The grammar and the program are dependent on the grammar formalism. The program, encapsulates the principles of the grammar formalism.

Organization of the Parser

One module of the parser takes care of morphology. For each word in the input sentence, a lexicon is looked up, and associated grammatical information is retrieved. The words are grouped together yielding nominals, verbals etc. Finally, the karaka relations among the elements are identified. This is shown in Fig. 2.2.

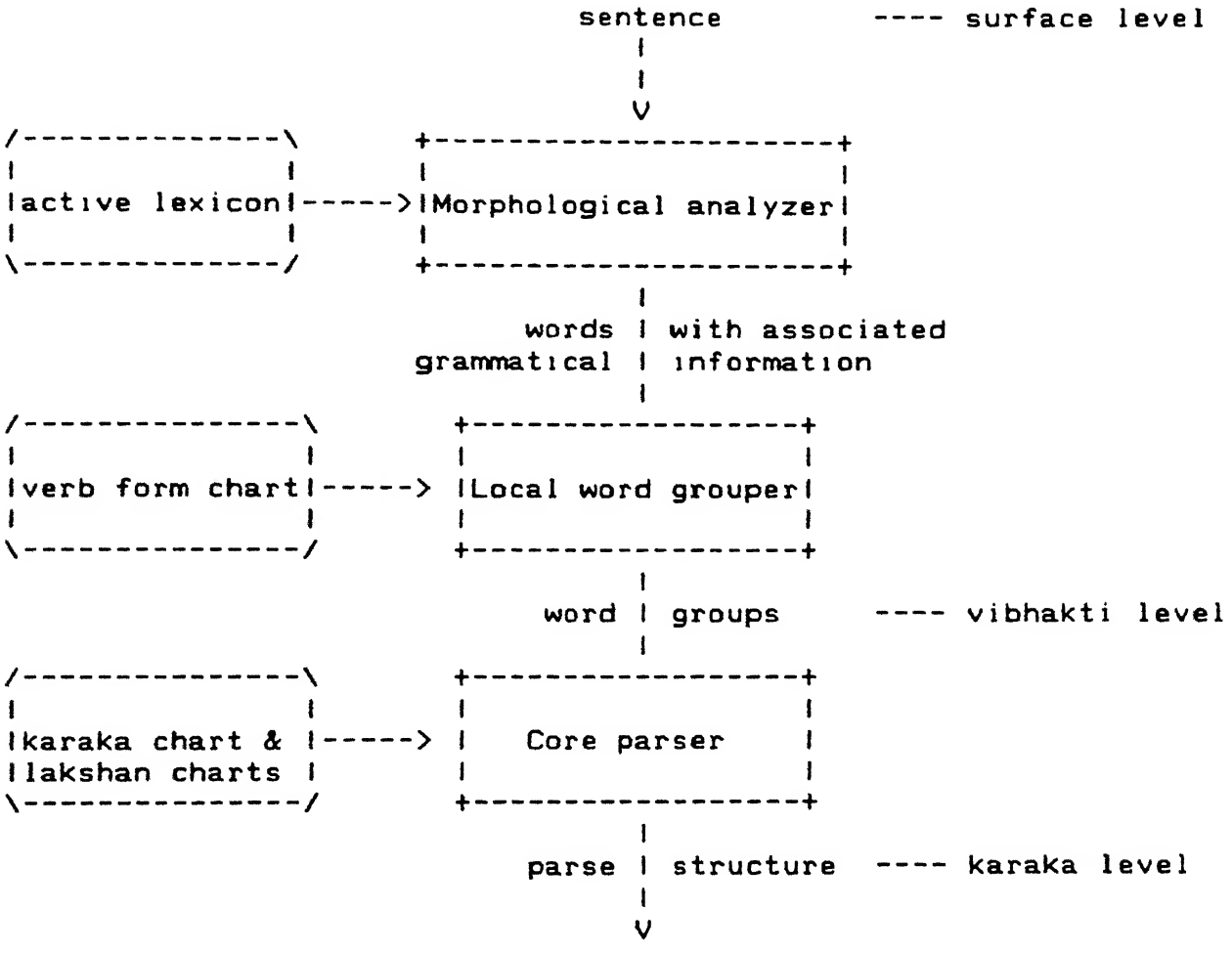


Fig. 2.2 Structure of the Parser

2.2.1 Morphological Analyzer

The function of this block is to take the source language sentence, provided as input to the parser and retrieve the lexical information associated with each word. In case of words with multiple meanings the analyzer returns the lexical information for each of them.

2.2.2 Local Word Grouper

Indian languages are relatively free in their word order, however, the word order is fixed in certain units. The units themselves may be moved in the sentence, without changing the broad meaning of the sentence. Examples of such units are the verbs followed by auxiliary verbs, nouns followed by postpositions. Groups once formed may not be split later, hence the groups are formed out of adjacent words, that are known to be unambiguous. The verb grouping is done based on the possible verb sequences that may occur and information about agreement. The noun groups are composed based on the form of the noun and the following postposition.

The phenomenon of local word grouping occurs to a greater extent in languages like Hindi where in the declension takes the form of a postposition and the auxiliary verbs are separated by word boundaries. In languages such as Telugu, local word grouping occurs to a lesser extent, since a word, on most occasions, is morphologically inflected.

2.2.3 Core Parser

Given the local word groups in a sentence, the core parser

- Identifies karaka relations among word groups
- Identifies senses of words

The first task requires knowledge of karaka-vibhakti

mapping, optionality of karakas, and transformation rules. The second task requires lakshan charts for nouns and verbs.

1. Identifying Karaka Relations

A structure called karaka chart stores information about karaka-vibhakti mapping and optionality of karakas. Initially, the default mapping is loaded into it for a given verb group in the sentence. Transformations are performed using the tam label. There is a separate karaka chart for each verb group in the sentence being processed. Information about semantic types of fillers of karaka roles is also available. But such information is limited to that necessary for removing ambiguity, if any, in karaka assignment.

For a given sentence after the word groups have been formed, karaka charts for the verb groups are created and each of the noun groups is tested against the karaka restrictions in each karaka chart (provided the noun group is to the left of the verb group whose karaka chart is being tested). When testing a noun group against a karaka restriction of a verb group, vibhakti information and semantic type are checked, and if found satisfactory, the noun group becomes a candidate for the karaka of the verb group. The verb groups are called demand groups, and the noun groups are called source groups.

2. Lakshan Charts for Sense Disambiguation

The second major task to be accomplished by the core

parser is disambiguation of word senses. This requires lakshan charts (or discrimination nets) for nouns and verbs. A lakshan chart for a verb allows identifies the sense of the verb in a sentence given its parse. Lakshan charts make use of the karakas of the verb in the sentence, for determining the verb sense.

Noun lakshan charts help disambiguate senses of nouns in a sentence. They make use of the parse structure (i.e., karaka relations) and the verb sense.

2.3 Extensions to the parser

The Paninian parser is extended to handle the problems discussed below.

2.3.1 Value Recognition and Unknown Terms

The traditional approaches to this problem are

1. List all the value terms in the lexicon
2. Search the database for the value terms (View the database as an extension of lexicon)
3. List all possible patterns of the value terms and take all unknown terms falling in this set to be value terms.

All the above approaches have disadvantages [BOL86]. The method we take is to restrict the user to enter the value terms in special markers. This restriction is

reasonable because we can integrate a menu for value term specification in future. It is not necessary that the entire interaction should be in natural language, it is only that the interaction should be natural.

The morphological analyser is modified to treat words appearing in special markers as nouns, and assign relevant default features. Since, these value terms can also play a karaka role, a default noun concept chart with semantic type T, is taken for them.

2.3.2 Pronouns and Definite Noun Phrases

The pronouns have a noun concept chart with all possible semantic types listed. The core parser is modified to treat pronouns also as source groups, while forming demand and source groups. For definite noun phrases, the determiner is identified and only the head is taken as a source word.

2.3.3 Ellipsis Handler

The core parser is modified to invoke the ellipsis handler in some of the cases when it fails to proceed.

2.3.4 Logical Division of Lexicon

The lexicon is divided into domain and non-domain parts. This information is useful in the mapping module.

CHAPTER 3

MAPPING

In this chapter we discuss the mapper and translator modules of the interface. The mapper module maps the parse structure to the intermediate representation while the translator module translates the intermediate representation to the SQL query. In Section 3.1 the representation of domain knowledge is discussed illustrating the representation for a library information system. The intermediate representation formed for the input question is given in Section 3.2. The procedure to map the parse structure to the intermediate representation is discussed in Section 3.3. The translation procedure from the intermediate structure to the SQL query is discussed in Section 3.4. The algorithms discussed do not handle full set of the problems in mapping and translation. The assumptions made are specified in each case.

3.1 Domain Knowledge

Domain knowledge is essential for a natural language interface (NLI) to achieve an acceptable level of performance. The domain knowledge that is needed is represented using Entity-Relationship (E-R) model and generalization hierarchy. They represent the minimal domain knowledge in a compact manner. This helps in making the task of system building easy and in achieving portability.

3.1.1 Knowledge Representation in Our System

We chose a library information system as an application area for the prototype NLI. In our system, domain knowledge is represented in an Entity-Relationship model.

E-R model consists of entities, attributes and relations between entities. It captures the functional dependencies between entities in a compact fashion. The E-R diagram for the library domain is shown in Fig. 3.1. The rectangle represents an entity set and the oval represents an attribute. An attribute is linked to its entity set by an undirected edge. Diamonds represent relationships. They are linked to their constituent entity sets by undirected edges. All the relationships shown in the Figure 3.1 are of the type many-many.

The E-R model for the domain is used as an information source independent of the modules using it. It is represented by a suitable data structure. A tool which allows the database administrator (DBA) to enter the E-R model for the domain interactively, can be built in future so that a different domain can be easily set up.

Taxonomies with a hierarchy of types and subtypes can be used to restrict the number of entities and relations. ISA links are used to show the generalization hierarchy. For example, there is an ISA link from manuals to books because a manual is a book.

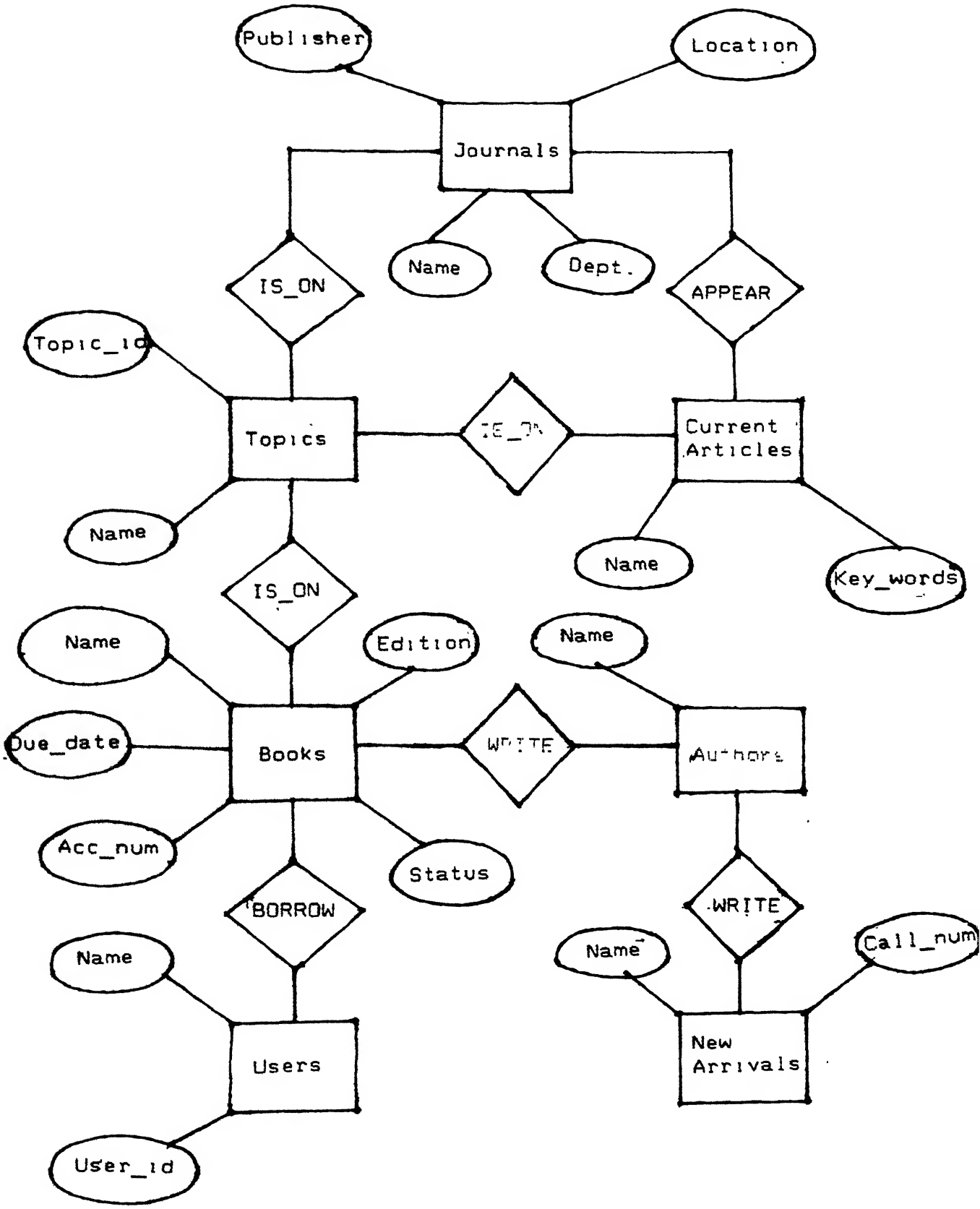


Figure 3.1 E-R Diagram for the Library Domain

3.2 Intermediate Representation

Intermediate representation is a representation of the user query obtained after analysis. It is based on Entity-Relationship model. The representation consists of nodes and directed arcs connecting the nodes. A node corresponds to the entity represented by a word concept, and the arcs to relationship between them as specified in the natural language question.

The structure of a node is as shown in Fig. 3.2.

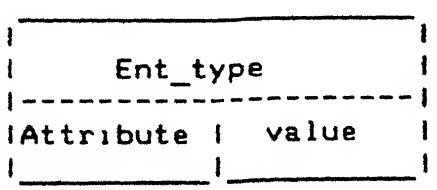


Figure 3.2. Structure of a Node.

The fields of a node are filled from the information provided by parse structure and map tables. The field Ent_type can be filled by the set of possible entity types for a word concept. The procedure for forming the intermediate representation is given in the next section.

A directed arc connects two nodes whose entity types have a relationship in the E-R model. The direction of the arc indicates the direction to be followed during code generation.

An example of an intermediate representation is shown in

Fig. 3.3. It corresponds to the question (the Roman coding scheme for the Hindi script is given in Appendix 2):

'NLP' nAmaka kitAba kA leKaka kOna hE?

'NLP' titled book -kA author who -pres?

Who is the author of the book titled 'NLP'?

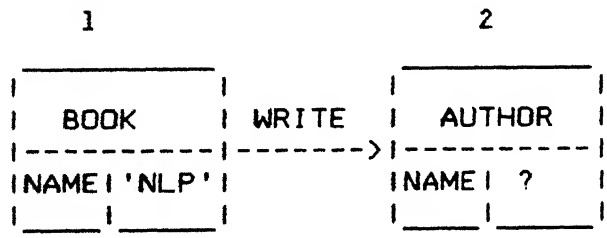


Figure 3.3 An Intermediate Representation

It shows two nodes node1 and node2 with entity types BOOK and AUTHOR respectively. Node1 has an attribute value pair (<NAME, 'NLP'>) while for node2 the attribute (NAME) is to be found. The directed arc connecting the nodes represents the relation WRITE.

The advantage of this intermediate representation is that we can easily ensure the semantic well-formedness of the question. Another advantage is that it is close to the natural language question, where database information is referred to in terms of entities and relationships in the world. This intermediate representation can serve as a starting point upon which we can build. For example, to represent quantifiers, new operators for set formation can be introduced.

3.3 Mapper

This module takes the parse structure and forms the E-R based intermediate representation. The map tables provide the information necessary for this mapping.

For example the word concept leKaka_1 (author) will have map table entry AUTHOR, an entity type, while the word concept likA_1 (to write) will have the map table entry WRITE, a relation with AUTHOR and BOOK being the entity types it relates in the E-R model.

The word concepts in the parse structure are mapped into one of the following types in intermediate representation.

Entity
Attribute
Relation
Value term
Quantifier
Time indicator
Place indicator

Complex sentences and sentences with multiple verb groups are not considered for mapping in our system. This means that the user is restricted to enter only simple sentences.

The mapper module consists of the following three phases :

1. The first phase forms the nodes of the intermediate structure. A set of all unresolved value terms in the parse structure is also obtained. The relations are stored for use in the second phase.

2. Making the intermediate structure fully formed : If the entity type referred to by a value term is unspecified, domain knowledge is used to find it. All the nodes which have more than one entity type in the Ent_type field are assigned a unique entity type in this phase. Finally, directed arcs connecting the nodes are formed.

3. Checking for well-formedness : A check is made to see whether the intermediate representation obtained is well-formed.

The three phases are explained in the following sections with examples.

3.3.1 First Phase

The first phase uses the procedure Get_ER_Info. The following example gives the function of the procedure.

'databases' nAmaka kitAba kisane liKI?

'databases' titled book who wrote?

Who wrote the book titled 'databases'?

The parse structure of this sentence is as follows :

```
verb : likA_1

karakas :
karwA       kOna_2
karma       kitAba_1
            shashti modifier  : 'databases'
            adjective modifier: nAmaka
```

The map table for the word concepts is shown in Figure 3.4.

Word concept	Type	Intermediate concept
'databases'	value term	'databases'
nAmaka	attribute	NAME
kitAba_1	entity	BOOK
kOna_2	entity	AUTHOR,USER
likA_1	relation	WRITE

Figure 3.4. A Sample Map Table.

The output of the procedure is shown below:

set of nodes:

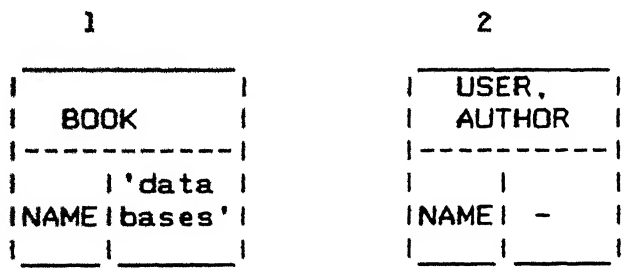


Figure 3.5. Set of Nodes Returned by Get_ER_Info.

Set of unresolved value terms : nil

Set of relations : WRITE

```
Procedure Get_ER_Info()
//
   This procedure
   a. Forms the nodes of intermediate structure
   b. Finds all unresolved value terms
   c. Finds all relations.
//

Input : parse structure and map tables.
Output: nodes, unresolved value terms and relations.

BEGIN
//In the first FOR loop the nodes are constructed//
FOR each word concept in the parse structure
Get the map table entry for the word concept
IF type is ENTITY THEN
        Mark the word 'entered'
        tmp_node <- get_new_node();
        //The procedure get_new_node returns a new empty node//
        tmp_node.Ent_type <- entity type from the map table entry
        IF the word has any modifiers THEN
    Get the map table entry for the modifier
        IF type is ATTRIBUTE THEN
                  tmp_node.attribute <- attribute
                  Mark the modifier 'entered'
        IF type is VALUE TERM THEN
                  tmp_node.value <- value term
                  Mark the modifier 'entered'
END_FOR

FOR each word concept in the parse structure
Get the map table entry for the word concept
IF type is ATTRIBUTE THEN
        IF the word is not marked 'entered'
        Mark the word 'entered'
        Get the map table entry for the shashti modifier of the wo
        IF the type of the modifier is ENTITY THEN
                nodex <- get_node_of(shashti modifier)
                // The node corresponding to the entity type
                of the modifier is returned //
                nodex.attribute <- attribute
IF type is VALUE TERM THEN
        IF the word is not marked 'entered'
                Mark the word 'entered'
                Enter it in a list of value terms
                Mark the value term as to be resolved later
IF type is RELATION THEN
        Mark the word 'entered'
        Insert the relation in a set of relations encountered
        //Used later to construct the arcs//
END_FOR
END
```

The execution of the algorithm for the example given above is as follows :

In the first FOR loop the nodes of the representation are formed. The first node formed is corresponding to the word concept kitAba_1. The entity type BOOK is assigned to the field Ent_type of the node. The modifiers of the word are nAmaka and 'databases'. The intermediate concepts of these modifiers are assigned to the fields attribute and value of the node respectively. The second node formed is corresponding to the word concept kOna_2. The entity types AUTHOR and USER are assigned to the field Ent_type of the node.

In the second FOR loop the relation WRITE, corresponding to the verb liKa_1 is entered in a set of relations encountered.

3.3.2 Second Phase

In this phase, three things are done: The entity types referred to by each unresolved value term are identified. All the nodes which have more than one entity type in the Ent_type field are assigned a unique entity type. Finally, directed arcs connecting the nodes are formed.

1. Value Term Disambiguation

Value term disambiguation can also be viewed as resolving deep level ellipsis. Deep level ellipsis is identified

if all the constituents to form SQL query are not available from the intermediate representation. Since, in this case the entity type referred to by a value term is missing it is identified as deep level ellipsis. This phase uses domain knowledge, captured in the E-R model, for disambiguation. Various techniques for disambiguation are

A. Using Verbs

Some of the verbs of the parse structure are mapped into relations. From the E-R model the entity types that are constituents of such a relation are obtained. The value term can refer to these entity types only.

For example in the following sentence

'OS concepts' kisane liKI?

'OS concepts' who-ne wrote?

Who wrote 'OS concepts'?

there is a verb liKa (to write). It relates the entity types author and book in the domain. The entity type book referred to by the value term 'OS concepts' is obtained from this knowledge.

B. The E-R model has information about the relation that each type of entity has with other types of entities. This information can be used in restricting ambiguity.

For example, in the following sentence :

'AI' kI kitAbeM kOna kOna sI hEM?

'AI' -kI books what -sI pres?

What are the books of 'AI'?

The value term 'AI' can refer to either an attribute of the entity type book or can refer potentially to all the other entity types in the E-R model. However, in the E-R model the entity type book is related to the entity types topic, user, and author only. Thus we can restrict the number of possible entity types the value term can refer to by using this knowledge. Here only a direct relationship between two types of entities is assumed.

C. If ambiguity is present even after the application of the above techniques then user interaction is needed.

After the entity type referred by the value term is found, a new node is created for the entity type. The value term is entered as the default attribute of the node.

2. Some of the nodes may have more than one entity type in the field Ent_type. This ambiguity can be resolved by using the techniques described in the previous section. All the nodes will have unique entity type after this resolution.

In the example given in Section 3.3.1, the node corresponding to the word concept kOna_2 (node 2), has two entity types AUTHOR and USER in the field Ent_type. The verb liKa_1 has the information to resolve it as AUTHOR because the

corresponding relation WRITE has only the entity type AUTHOR in its constituent entity types.

3. Directed Arc Construction

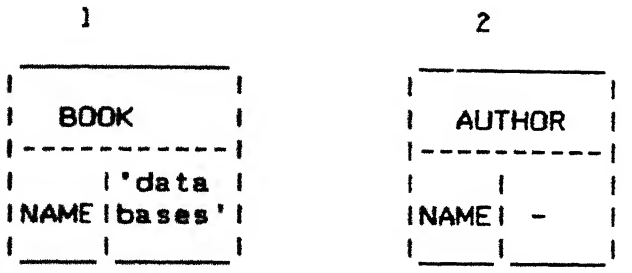
The nodes are connected by directed arcs in this phase. The entity types of the nodes connected by a directed arc have a relationship in the E-R model. The directed arc represents that relation. The direction of the arc is used in the translator module. If a directed arc is from node A to node B, then during translation, the query for node A is to be generated first to be used in forming the query of node B later.

We define a value node as a node for which all the attributes have a value specified and a non-value node as a node for which an attribute has unspecified value. It is assumed that quantifier scoping is not necessary for the input question and that all the relationships between nodes are direct. Thus where scopes of quantifiers are important, the NLI fails to produce an appropriate formal query. Direct relationship between two nodes means that in the E-R model there is a single relation connecting the entity types of the nodes.

This phase uses the procedure Construct_arcs. We continue with the example of the Section 3.3.1 to show the functioning of the algorithm.

Input

Set of nodes:



Set of relations : WRITE

Output(Intermediate structure) :

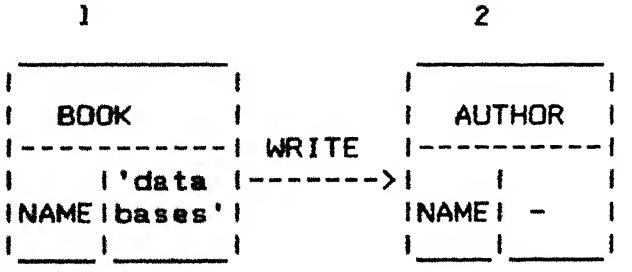


Figure 3.6. Output of Construct_arcs

```
Procedure Construct_Arcs()
//
   This procedure forms an intermediate structure
   containing the nodes and directed arcs between
   the nodes.
//
Input : set of nodes,
        parse structure with node information,
        and set of relations.
Output: Intermediate structure.

BEGIN
IF only a single node is present in the set of nodes
        Include it in the intermediate structure.
ELSE
IF both a modifier and a modificand in the
parse structure are of type ENTITY THEN
        node1 <- node_of(modifier)
        // The procedure node_of gives the node formed for the word
        node2 <- node_of(modificand)
```

```
        IF Are_related(node1.Ent_type,node2.Ent_type) THEN
        //Check to see if E-R model permits this relation//
          Make a directed arc from node1 to node2
          and label it by the direct relation.
END_FOR
FOR each relation in the set of relations
        Get the entity tuples it relates.
        IF the nodes corresponding to the
    two entity types of a tuple are found THEN
    IF one is a value node and other is a non-value node
    Make a directed arc from the value node to the non-value node
        ELSE IF both are value nodes
        Make a directed arc such that no node will have more than
        one outgoing arc in the resulting structure.
             IF the arc can not be made
                print Error message
        ELSE IF both are non-value nodes
                print Error message
        //The case of both being non-value nodes is not considered
          and results in error//
END_FOR
END
```

The execution of the procedure is as follows: The relation WRITE relates the entity types book and author. The two nodes with these entity types are node1 and node2 respectively. Node1 is a value node while node2 is a non-value node. So, a directed arc from node1 to node2 is formed and it is labeled by the relation WRITE. The output structure is shown in Fig. 3.6.

3.3.3 Third Phase

The module takes the intermediate structure formed in the previous phase as its input, and checks if a valid SQL query can be formed. The checks are

The intermediate structure should contain at least one node.

Every node should have at least one incoming arc and at most one outgoing arc.

No cycle is present in the intermediate structure.

All the attributes of a node should be valid for the entity type specified for the node.

Each value of a node should have an associated attribute.

More than one attribute of a node can not have an unspecified value.

For example, in the following question

'SIGART' nAmaka jarnal kA leKaka kOna hE?
'SIGART' titled journal -kA author who pres?
Who is the author of the journal titled 'SIGART'?

The two nodes that are formed with entity types JOURNAL and AUTHOR corresponding to the word concepts jarnal_1 and leKaka_1 have no relation and hence they are not connected. This is detected in this module and suitable error message is given.

The example of Section 3.3.1 passes this well-formedness test.

3.4 Translator

The intermediate structure that passes the well-

formedness test will be a directed tree because

a. Every node has a directed arc.

b. No cycle is present.

c. No node has more than one outgoing arc.

The node with no outgoing arcs is designated as the root of the directed tree. The direction of the arcs is from a child node to its parent node.

The translator module takes the directed tree and generates the semantically equivalent SQL query. Translator tables, which give for each attribute of an entity type its corresponding field and record type in the database schema, are used for this purpose. These tables also contain the join conditions necessary to connect two record types.

3.4.1 Target Language

Since we considered only relational DBMS, SQL is the natural choice for the target language. SQL is a powerful non-procedural language. We considered only a subset of the SQL language. The syntax of commands in SQL is given in Appendix 1.

Basically, an SQL query consists of clauses, each of which is preceded by a keyword. The 'select' clause introduces every query. This clause tells which fields are to be selected. The 'from' clause is also required for a valid query. This clause tells which record type the fields are to

come from. There is an optional 'where' clause which allows to select records based on the results of a boolean expression. So, the translator should try to construct these three clauses to form the SQL query.

3.4.2 Code Generation for Directed Tree

The procedure Tree_code_gen is used to form the SQL query for the tree. This procedure generates the query by combining the query of each node of the tree. The procedure Node_gen described in the next section is used for getting the query of a node.

We continue with the example of the Section 3.3.1. The input to this procedure is the directed tree of Fig. 3.6. The query for the node1 (BOOK) is generated first as it is a leaf node. While forming the query of node2 (AUTHOR) the query attached to node1 is used. The translator tables used for in Node_gen are shown in Fig. 3.7.

Attribute	entity type	record type	field
NAME	BOOK	rt1	title
NAME	AUTHOR	rt2	author

Figure 3.7. A Sample Translator Table.

The first entry in the table gives the record type and field for the attribute NAME of node1 (BOOK). This information is used to construct the query for it. The query for

node2 is constructed similarly except that it uses the query formed for node1 also. The final query formed is shown in Fig. 3.8.

```
select rt2.author
from   rt2
where  rt2.accnum =
                    select rt1.accno
                    from rt1
                    where rt1.title = 'NLP'
```

Figure 3.8. Generated SQL Query.

```
Procedure Tree_code_gen()
//
   This procedure takes a directed tree
   and generates the semantically equivalent
   SQL query.
//
Input : A directed tree.
Output: Equivalent SQL query.

BEGIN
Get the root node of the directed tree
//Root node has no outgoing arcs//
start <- root node
CALL BFT(start);
// A breadth first traversal of the tree is carried out
   beginning at the root node. The number of levels in
   the tree is returned.
//
FOR i <- number_of_levels down to 1 do
        FOR each node at level i do
        CALL Node_gen();
        // This procedure returns the SELECT list, FROM list
           and CONDITION list for the node
        //
        FOR each child of this node do
                Get the sub-query attached to the child
                Get the join conditions with the record type of the chi
                Form a condition using the sub-query and join condition
                Add this condition to the CONDITION list of the node
        END_FOR
        CALL form_query()
        // This procedure takes the SELECT list, FROM list and
           CONDITION list of the node and returns the SQL query
        //
        Attach the formed query to the node
        END_FOR   // Inner for loop//
```

```
END_FOR       // Outer for loop//
END
```

The working of the procedure is described using the tree formed for the example taken. The root node of this tree has the entity type AUTHOR while its child has the entity type BOOK.

Call to procedure BFT returns the number of levels as two. The procedure Node_gen is called first with node1 (BOOK) as its argument. The following lists are returned for this node.

```
SELECT list      rt1.accno
FROM list        rt1
CONDITION list   rt1.title='NLP'
```

The call to the procedure form_query() returns the following SQL query.

```
select rt1.accno
from   rt1
where  rt1.title='NLP'
```

The procedure Node_gen is called next with node2 as its argument. The lists returned for this node are

```
SELECT list     rt2.author
FROM list       rt2
CONDITION list  (null)
```

The child of this root node has a sub-query formed in the previous iteration. The record types rt2 and rt1 are joined

on the fields accnum and accno respectively. The final query formed is shown Fig. 3.8.

The resulting SQL query may not be optimal from the database retrieval view-point. A query optimizer can be used on the generated query before passing it to the DBMS.

3.4.3 Code Generation for a Node

The procedure Node_gen uses the translator tables to give for a node the clauses needed in the SQL query.

```
Procedure  Node_gen()
// This procedure gives for a single node of the tree
   the clauses needed for SQL query
//
Input : A single node of the tree and translator tables.
Output: SELECT list, FROM list and CONDITION list.

BEGIN
IF no attributes are specified for the node
        Assume key attribute of the entity type
FOR each <attribute,value> pair of the node
        Get the record type and the field of the attribute
        Include the record type in the FROM list
        Update the CONDITION list
END_FOR
IF all attributes have value terms specified
        For the SELECT list assume key attribute is asked
        Get the record type and the field of the key attribute
        Include the record type in the FROM list
        Form a SELECT list
IF any attribute whose value is unknown is present
        Get the record type and the field of the attribute
        Include the record type in the FROM list
        Form a SELECT list
IF FROM list has more than one record type
        FOR each pair of record types
        cond <- join condition of the two record types
        CONDITION list <- CONDITION list and cond
END
```

For the example in Section 3.3.2, this procedure is

called for the node with entity type BOOK first, and for the node with entity type AUTHOR later. The SELECT list, FROM list and CONDITION list returned for each node is given in the previous section.

3.5 Implementation

All the algorithms described for the mapper module and the translator module have been implemented in the prototype NLI. As previously mentioned, complex sentences and sentences with multiple verb groups are not considered. However, the algorithms require very few changes to handle these cases.

3.6 Accessing the Database

A sample database for the library domain is created on HCL Horizon-3 machine using the relational DBMS, UNIBASE. The NLI system runs on a remote machine. It collects the user input and the SQL query obtained is stored in a file. The host machine (Horizon-3: where the database resides) is accessed using the network to execute the query remotely. The answer is stored in another file. The answer file is processed to implement numeric quantifiers and to inform the user in case of NULL answers. A sample run for a question is shown in Appendix 3.

CHAPTER 4

REFERENCE AND ELLIPSIS

To support natural interaction it is desirable to allow the use of anaphoric reference and elliptical constructions across sentence sequences. Reference and ellipsis are very hard problems to attempt in a general context. The restricted domain of discourse that is defined by the contents of the database makes it possible to address these problems in our case.

4.1 Ellipsis

A question is called elliptic if one or more of its constituents are omitted. It is important for an NLI to handle ellipsis for brevity in communication.

There are two types of ellipses:

1. Surface Level Ellipsis: This is detected and handled at syntactic level.
2. Deep Level Ellipsis: This is identified and handled during intermediate structure construction.

4.1.1 Surface Level Ellipsis

Surface level ellipsis is handled in the parser, using pure syntax-oriented approaches. The parse structure information is sufficient to handle this type of ellipsis. The processing of a surface level elliptical fragment involves the following steps :

1. Recognize input as elliptical

2. Get source in antecedent

3. Construct new full utterance

A. Recognition

The user input is recognized as elliptical at the surface level, if either one or more of the following cases occur in the parsing phase:

- No demand word is present
- A mandatory karaka is absent
- A Modifier has no head

We make an assumption for surface level ellipsis that the elliptical fragment corresponds in parse structure to that of the previous question and there is no syntax violation in the question. The parser is modified so that instead of failing for the above cases, it invokes an ellipsis handler. The parser continues with its task after ellipsis handler returns control. This means that if more than one of the above cases occur in a fragment, the parser invokes the ellipsis handler as many times.

B. Ellipsis Handler

For each of the above three cases, the ellipsis handler is invoked by the parser. We describe below the process of

identifying the source and instantiation of new full utterance for each case.

Case 1: Demand Word is Absent

. From the previous parse structure take the demand word.

. Continue with the parsing using the new demand word.

For example, in the following dialogue:

'NLP' nAmaka kitAba kisane liKI hE?
'NLP' titled book who-ne wrote pres?
Who wrote the book titled 'NLP'?

'DBMS' nAmaka kitAba?
'DBMS' titled book?
The book titled 'DBMS'?

The second question has no demand word, so it takes the demand word liKa (to write) from the previous question.

Case 2: Mandatory Karaka is Absent

. In the previous parse structure identify the word with the same karaka role.

. Continue with the parsing using the word as new karaka.

For example, in the dialogue given in case 1, for the second question karwA karaka is absent, so it takes the word kisane (who) which has the same role (i.e., karwA) from the previous question. Note that for this example, the ellipsis

handler is invoked for the second time, the first time being for demand word absence.

Case 3: A Modifier has no Head

. From the previous parse structure try to find the head for the modifier

. Assign new karaka role for the head word

. Continue with the parsing using the new head

For example, in the following dialogue:

'IEEE' ke jarnal kOna kOna se hEM?
'IEEE' -ke journals what -se pres?
What are the journals of 'IEEE'?

'ACM' ke kOna kOna se hEM?
'ACM' -ke what -se pres?
That of 'ACM'?

In the second question there is no head word for the shashti group ('ACM' ke), so head word assignment procedure is applied for this shashti group with the previous parse structure as the other argument. In this example, jarnal (journals) is the head found.

4.1.2 Deep Level Ellipsis

Deep level ellipsis can be detected only while forming the meaning of the question. The user input is recognized as

elliptical at the deep level, if all the constituents needed to form the SQL query can not be obtained from the intermediate representation of the question. Various methods for resolution of deep level ellipsis are:

- Use domain knowledge
- Use previous intermediate representation, if the previous parse structure corresponds to current structure
- User interaction

The use of domain knowledge for resolution of deep level ellipsis has been discussed in Chapter 3.

4.2 History List Formation

The main technique for handling intersentential reference is the maintenance of a record of all objects mentioned in the preceding sentence, called history list. For each object the syntactic and semantic features associated with it are stored. We use the elements in this list in finding the referent of the anaphor.

We put a restriction that the user can refer back to the entities of the previous sentence only, using anaphoric reference. Thus it is enough if we store the objects present in the previous sentence. This implies that the user can not use anaphora in the first sentence.

4.2.1 Algorithm

The formation of history list is done after the anaphora resolution is over for the current sentence.

```
Procedure Form_hlist()
//This procedure constructs the history list
  from the parse structure of the sentence.
//
Input: parse structure and antecedents if any.
Output: History list.

BEGIN
FOR each of the word groups in the parse structure
IF it is marked as anaphor
THEN
        Get the referent attached to the anaphor
        Add this object to the history list
ELSE IF it has a karaka role or is a shashti modifier
        THEN
        Create a new object
        Attach relevant syntactic and semantic features to the object
    Add this object to the history list
END_FOR
END
```

4.3 Reference

We consider the case of intersentential anaphoric reference only. In this type of reference only the objects already present in the conversational context are identified.

4.3.1 Anaphora

The term 'anaphora' refers to reflexive pronouns, general pronouns, definite noun phrases, etc. Anaphora resolution involves finding referents of these in a discourse which may consist of more than one sentence. The tests designed to handle the intersentential anaphora can be used for handling intrasentential anaphora as well. The detection

of anaphora is done in the parser module and the resolution is done later using the parse structure obtained from the parser module. We need consider only anya purusha (third person) pronouns for our system as the other types of pronouns, uttama purusha (first person) pronouns and madhyama purusha (second person) pronouns, refer to the user and the NLI system respectively and hence do not fall under anaphoric reference. We do not consider the scope of reference because the quantifier scope determination is not done.

a. Detection: This is a simple task involving marking of all the anya purusha pronouns in the parse structure.

b. Resolution: Designing an anaphora resolution system is difficult because there exists no single, coherent theory upon which to build. However there are many partial theories, each of which accounts for a subset of the phenomena that influence the use and interpretation of anaphora. These can be designed as a set of tests each of which acts as a filter. A filter takes a set of possible candidates as its input and gives those candidates that pass the test as its output. We describe below each test and illustrate how it acts as a filter.

1. Recency

This means that the history list is searched starting from the most recently mentioned objects, until a suitable candidate is found. This is not of much use in our system,

as only a single sentence history list is maintained.

2. GNP Agreement Test

This test filters all those candidates not in agreement with the number and person of the anaphor. For example in the following dialogue:

'AI' nAmaka viSaya para kOna kOna se jarnal hEM?
'AI' named topic on what -se journals pres?
What are the journals on the topic named 'AI'?

unameM kOna kOna se Artikals hEM?
They-meM what -se articles pres?
What are the articles in them?

The anaphor ve (root of unameM, meaning they) has jarnal (journals) as referent as the other candidate viSaya (topic) is filtered due to number agreement failure. This test is language dependent (this is for Hindi).

3. Semantic Type Consistency

The semantic type associated with the anaphor is tested for consistency with the semantic type of each candidate and those candidates found inconsistent are filtered. If the anaphor plays a karaka role, we get the semantic type of it, from the karaka chart of the main verb. For example, in the following dialogue:

'NLP' nAmaka kitAba kA leKaka kOna hE?

'NLP' titled book -kA author who pres?

Who is the author of the book titled 'NLP'?

usane 'AI' para kOna sI kitAbeM liKI?

He-ne 'AI' on what -sI books wrote?

What books did he write on 'AI'?

The anaphor vaha (root of usane, meaning he) is assigned the semantic type manuSya (human) from the karaka chart of the main verb liKa (to write) as it plays a karaka role. This is used to select leKaka (author) as the referent, as the other candidate kitAba (book) is filtered, since it has a semantic type BOwika_others (other physical), inconsistent with manuSya (human).

4. Domain Model Test

This test uses knowledge about the structure of the domain (supplied by the E-R model). This test uses the observation that the shashti modifier and its head are related and if the head is a domain word, the referent of the anaphor should have a relation with the head in the domain. For example in the following dialogue:

'NLP' nAmaka kitAba kA leKaka kOna hE?

'NLP' titled book -kA author who pres?

Who is the author of the book titled 'NLP'?

usakA nambar kyA hE?

He/it-kA number what pres?

What is his/its number?

The anaphor vaha (he/it) is a shashti modifier of nambar (number), so they are related. As in our domain (library information system) nambar (number) can only be related to kitAba (book) it is taken as the referent, filtering leKaka (author).

5. User Interaction

If the above listed filters fail to give the referent, we ask the user to select it from the set of candidates using a menu. While taking a particular karaka role as default referent is a good choice, it doesn't work always. So it is safe to go for user interaction. For example in the following dialogue:

saBI tApiks ke kOna kOna se jarnal hEM?
All topics -ke what -se journals pres?
What are the journals of all topics?

unake Artikals dikAo?
They-ke articles show?
Show their articles?

The anaphor ve (root of unake, meaning their) has both jarnal (journals) and tApiks (topics) as candidates, even after the filters are applied.

Definite Noun Phrases

The detection of definite noun phrases is done by using the determiner used in the phrase. The resolution methods use the same tests described above, except that the semantic type consistency test can use the semantic type of the noun itself. For example, in the following dialogue:

'NLP' nAmaka kitAba kisane lI hE ?
'NLP' titled book who took pres?
Who took the book titled 'NLP'?

usa kitAba kA nambar kyA hE ?
That book -kA number what pres?
What is the number of that book?

The determiner is usa (that) and the head kitAba (book) is used as anaphor.

There is a non-anaphoric use of definite noun phrases called deictic reference. Deictic reference refers to the time and place of an utterance. For example, the noun group: isa mAha (this month).

4.3.2 Others

Noun groups such as "anya kitAbeM" (other books), where we have to identify a set of objects, disjoint from the set of objects referred in the previous sentence, are handled. We make use of the SQL operators available to handle these. For this example we use the NOT operator provided in SQL.

4.4 Domain Knowledge in Resolution

In the resolution methods described above for anaphora and deep level ellipsis some amount of domain knowledge is used. It is important to use the application-specific knowledge, because general linguistic analysis alone can not lead to complete formation of the meaning of the question. A detailed description of the use of domain knowledge for deep level ellipsis is given in Chapter 3.

CHAPTER 5
CONCLUSIONS

The issues of reference and ellipsis, in the context of a natural language interface to databases, are addressed in the present work. E-R model is used to capture the structure of the database. A database independent intermediate representation of the question is used in mapping algorithms. The issue of transportability is also addressed.

Since reference, ellipsis and other phenomena occur frequently in dialogues involving natural language interfaces to databases, it is important to handle them. These issues do not have adequate solutions in a general context. But, it is possible to handle them in a restricted domain as in our case. The tests developed for handling inter sentential anaphora can be used for intra sentential case as well. The Paninian parser is modified to invoke surface level ellipsis handler in some of the cases when it finds the sentence to be incomplete. Deep level ellipsis is handled while forming the intermediate representation of the question. A single sentence history list is adequate, as longer back references are not frequent in our case. In fact, the sample input collected does not contain any reference back by more than one sentence.

An added advantage of using the Paninian parser is its adaptability to all Indian languages. The E-R model acts as

an information source detached from the modules using it. As the E-R model is easy to setup for a database, it helps achieve easy transportability to a new application. A database independent representation of the meaning of the query is formed before constructing the formal database query. This separation is useful in making the NLI transportable across different databases. It also ensures that the advances in general purpose parsers are readily available to the NLI.

Further Extensions

The prototype NLI can be extended to handle some of the problems described below.

1. In the methods to handle surface level ellipsis, it is assumed that the previous parse structure corresponds to the current structure and that there is no syntax violation. Methods to handle ellipsis when these assumptions do not hold can be explored.

2. In case of a null answer to the question it is desirable to inform the user more. Providing useful responses involves testing the presuppositions in the question.

3. **Quantifiers:** For quantifiers, the problem of scope determination needs to be addressed. Another problem is to represent them suitably in the intermediate representation. Some new operators may be introduced for this purpose. As SQL is a non-procedural language, the operators forming

subsets in the intermediate representation won't have any directly corresponding operators in SQL. So, some transformations may be necessary, to use the operators present in SQL.

4. The system can be extended to be tolerant of user errors. Such errors might pertain to spellings, sentence constructions, agreement rules, etc. Correction by the system should be implicit without prompting it to the user. Other kinds of errors pertaining to misconceptions are to be detected and corrected explicitly.

REFERENCES

[ALL87] Allen, James, Natural Language Understanding, Benjamin Cummings , Menlo Park, 1987.

[BHA89] Bhanumati, B., An Approach to Machine Translation among Indian Languages, Tech. Report TRCS-89-90, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, Dec. 1989.

[BHR90a] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Computational Grammar for Indian Languages Processing, Tech. Report TRCS-90-96, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, Feb. 1990a.

[BHR90b] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Computational Framework for Indian Languages, Course Notes for Intensive Course in NLP, Vol. 1, Techincal Report, TRCS-90-100, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, July 1990b.

[BHR91] Bharati, Akshar, Vineet Chaitanya, B.N.Patnaik, and Rajeev Sangal, Meaning and Natural Language Processing, Tech. Report TRCS-91-116, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, Mar. 1991.

[BOL86] Bolc, L., and Jarke, M.(eds), Cooperative Interfaces To Information Systems, Springer-verlag, 1986.

[CAR83] Carbonell, J.G., Discourse Pragmatics and Ellipsis Resolution in Task-oriented Natural Language Interfeces, in proceedings of the 23rd meeting of ACL, 1983.

[DAT86] Date, C.J., An Introduction To Database Systems, Addison-Wesley, Reading Mass., 1986.

[FRE86] Frederking, R.E., Natural Language Dialogue in an Integrated Computational model, Ph.D thesis, Dept. of Computer Sc. and Engg., ,Carnegie-Mellon Univ., CMU-CS-86-178.

[GRO86] Grosz, Barbara J., et. al. (eds), Readings in Natural Language Processing, Morgan Kaufmann, California, 1986.

[JON86] Jones, Karen S., Robust, Cooperative and Transportable Natural Language Front ends to Databases, Computer science & Informatics, J. of Computer Society of India, vol.18 no.2.

[RIC88] Rich, E., and Susann, L., An Architecture for Anaphora Resolution, in proceedings of the Second Conference on applied Natural Language Processing, ACL, Feb., 1988.

[SOW90] SowA, J.F., "Knowledge Representation in Databases, Expert systems, and Natural Language" in Artficial Intelligence In Databases and Information Systems, ed. by Meersman R.A, et. al., pp 17-43, North-Holland, 1990.

[SRI90] Srinivas, B., Linguist's Workbench: A Grammer Development Tool for Indian Languages, M.Tech. thesis, Dept. of Computer Sc. and Engg., I.I.T. Kanpur, May 1991.

[ULL91] Ullman, J.D., Principles Of Database Systems, Galgotia Publications, New Delhi, 1991.

[ZOE83] Zoeppritz, M., Human Factors of a Natural Language Enduser System in End User Systems and Their Human Factors, ed. by Blaser, A., and Zoeppritz, M., Lecture Notes in Computer Science, pp 62-93, Springer-verlag, 1983.

APPENDIX 1

STRUCTURED QUERY LANGUAGE (SQL)

The subset of the SQL keywords used in our system is as follows:

and	in	not	or
from	is	unique	select
where	count		

A brief description of some keywords is given.

-- select

The `select' clause introduces every query. It has the following form:

select field or expression [field or expression, ...]

--- from

The `from' clause tells which record type the fields are to come from.

from record type [record type, ...]

This clause is used in conjunction with the `select' clause. Fields to be selected with `select' must be contained the files listed in the `from' clause.

--- where

The `where' clause allows you to select or reject records based on the results of a boolean expression.

--- and

The standard boolean operator `and' can be used to connect simple boolean expressions to form more complex expressions. Square brackets are used to indicate which part of the expression will be evaluated first.

--- or

The standard boolean operator `or' can be used to connect simple boolean expressions to form more complex expressions.

--- not

The `not' keyword is used as part of the `where' clause to negate an entire boolean expression.

--- unique

If a query does not select a primary key field from one of the record types, it is possible for that query to produce rows that are exact duplicates of each other. Sometimes, these duplicates are not desired. The `unique' operator is provided to let you suppress duplicate rows in a query result.

APPENDIX 2

Roman coding scheme for Indian scripts

a	A	i	I	u	U	q	e	E	o	O	M	H	z
अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ	ं	ः	ँ

k	K	g	G	f
क	ख	ग	घ	ङ
c	C	j	J	F
च	छ	ज	झ	ञ
t	T	d	D	N
ट	ठ	ड	ढ	ण
w	W	x	X	n
त	थ	द	ध	न
p	P	b	B	m
प	फ	ब	भ	म
y	r	l	v	
य	र	ल	व	
S	R	s	h	
श	ष	स	ह	

Examples:

rAma	kqRNa	jFAna	Sawru	AzKa	yakRa
राम	कृष्ण	ज्ञान	शत्रु	आँख	यक्ष

To enhance readability, some changes are made to the coding scheme in this Thesis. They are (Instead of the first letter on the left of the arrow, the letter to the right is used):

R -> S w -> t W -> T x -> d X -> D

APPENDIX 3

Sample run

A sample run of the NLI is shown below. The output shows the stages in the processing of the question. The Roman coding scheme of the Hindi question has been modified as specified in Appendix 2 for readability.

```
ENTER QUESTION : 'OS concepts' nAmaka kitAba kisane lI hE?

OUTPUT OF THE PARSER y/n? y

|          karaka    |root      |concpt     |positn     |gen    |num   |
|kartA               |kOna      |kOna_2     |4          |any    |one   |
|karma               |kitAba    |kitAba_1   |3          |f      |one   |
-------------------------------------------------------------------------
|verb_root           |concpt    |positn     |tam                       |
|le                  |le_1      |5          |yA_hE                     |
-------------------------------------------------------------------------

HISTORY LIST y/n? y

        2 CANDIDATES FORMED

        CANDIDATE 1
                ng
                kitAba
                f
                one
                BOtika_others

        CANDIDATE 2
                ng
                kOna
                any
                one
                manuSya

INTERMEDIATE REPRESENTATION y/n?y

        The number of nodes in the
        intermediate representation are: 2

        BOOK
                Name -- OS concepts
```

USER

Name -- ?

Relation is: BORROW

THIS IS THE SQL QUERY :

```
select rt3.user_name
from rt1,rt3
where rt1.title = 'OS concepts' and rt1.accno = rt3.accno
```

ACCESSING THE DATABASE. PLEASE WAIT...

REPLY>

K. Srinivas

ENTER QUESTION : 'Neural nets' nAmaka kitAba kA leKaka kOna hE

OUTPUT OF THE PARSER y/n? y

karaka	root	concpt	positn	gen	num
kartA	leKaka	leKaka_1	4	m	one
kartA_sam_aDi	kOna	kOna_2	5	any	any

verb_root	concpt	positn	tam
hE	hE_2	6	hE

HISTORY LIST y/n? y

2 CANDIDATES FORMED

CANDIDATE 1

sh_g
kitAba
f
one
BOtika_others

CANDIDATE 2

ng
leKaka
m
one
manuSya

INTERMEDIATE REPRESENTATION y/n?y

```
        The number of nodes in the
        intermediate representation are: 2

        BOOK
                Name -- Neural nets

        AUTHOR
                Name --  ?

        Relation is: WRITE

THIS IS THE SQL QUERY :

select rtl.author
from rtl
where rtl.title = 'Neural nets'


ACCESSING THE DATABASE. PLEASE WAIT...

REPLY>
                Minsky, M.


ENTER QUESTION : usakI anya kitAbeM kOna kOna sI hEM

OUTPUT OF THE PARSER y/n? y

-------------------------------------------------------------------
|          karaka    |root       |concpt     |positn    |gen   |num   |
|kartA               |kOna        |kOna_2     |4         |any   |any   |
|kartA_sam_aDi       |kitAba      |kitAba_1   |3         |f     |many  |
-------------------------------------------------------------------
|verb_root           |concpt      |positn     |tam                     |
|hE                  |hE_3        |7          |hE                      |
-------------------------------------------------------------------


ANAPHORA RESOLUTION

        AFTER DOMAIN MODEL TEST --

        Anaphor    : vaha
        Antecedent: leKaka


        HISTORY LIST y/n? y

        2 CANDIDATES FORMED

        CANDIDATE 1
                ng
                leKaka
```

```
                m
                one
                manuSya

        CANDIDATE 2
                ng
                kıtAba
                f
                many
                BOtıka_others


INTERMEDIATE REPRESENTATION y/n?y

        The number of nodes ın the
        ıntermedıate representatıon are: 2

        AUTHOR
                Name --  *    //poınter to Antecedent//

        BOOK
                Name --  ?

        Relatıon ıs: WRITE


THIS IS THE SQL QUERY :

select rtl.tıtle
from rtl
where not[rtl.title = 'Neural nets'] and rtl.author =
          select rtl.author
          from rtl
          where rtl.title = 'Neural nets'


ACCESSING THE DATABASE. PLEASE WAIT...

REPLY>
         AI in the 80's
```